REMARKS/ARGUMENTS

The Examiner and the Supervisory Examiner are thanked for granting the Applicant an in-person interview on August 3, 2004. During the Interview, several distinguishable features of the claimed invention were discussed (Summary of the arguments presented are reproduced below for the Examiner's convenience). Nevertheless, agreement was reached that the Applicant will further clarify the subject matter regarded as the invention solely in order to expedite prosecution. Accordingly, the applicant has further clarified the claims. The Applicant, however, respectfully reiterates the argument previously submitted and respectfully submits that the claimed invention is patentable over the cited art.

In the Final Office Action, the Examiner has rejected claims 1-22 under 35 U.S.C. §102 as being anticipated by U.S. Patent No. 6,081,665 (*Nilsen et al.*). This rejection is fully traversed below.

Initially, it is respectfully submitted that a job can be started on a virtual machine for each application (see, for example, Specification page 1, lines 33, 34). However, in conventional systems, when more than one application is to be executed concurrently, more than one virtual machine must be started such that each application has its own instance of a virtual machine (Specification, page 1, lines 30-33). The claimed invention, however, relates to a virtual machine that can support two or more jobs that can be associated with two or more applications (see, for example, claim 1).

The Applicant respectfully reiterates the arguments submitted in the amendment dated February 10, 2004, and respectfully submits that *Nilsen et al.* does not pertain to a virtual machine that can support two or more jobs can be associated with two or more applications. Again, it is noted that *Nilsen et al.* pertains to a method for efficient soft real-time execution of portable byte-code computer programs. To this end, *Nilsen et al.* describes a single abstract virtual machine execution stack with multiple independent stacks in order to improve the efficiency of distinguishing memory pointers from non-pointers (*Nilsen et al.*, Abstract). However, it is respectfully submitted that *Nilsen et al.* does NOT teach or suggest a virtual machine arranged to enable two or more jobs to run.

Although it is earnestly believed that distinction between *Nilsen et al.* and the claimed invention is apparent, solely for the Examiner's convenience, some of the features of the claimed invention that are neither taught nor suggested by *Nilsen et al.* are discussed below.

***(a) Nilsen et al.* does NOT teach or suggest a virtual machine arranged to enable two or more jobs, which can be associated with two or more applications, to run thereon (claims 1, 6, 13, 19 and 22)**

It is noted that *Nilsen et al.* states that Java is a threaded programming language, meaning that multiple flows of execution may be active concurrently, and all thread can have access to the same global memory pool (*Nilsen et al.,* col. 5, lines 39-42). However, it is respectfully submitted that the knowledge that Java is a threaded programming language does NOT teach or remotely suggest a virtual machine arranged to enable two or more jobs, which can be associated with two or more applications, to run thereon.

It is further noted that *Nilsen et al.* states that an instruction first consults an object's lock field. If this field is NULL, it allocates a HashLock object, initializes its count field to 1, sets it u.owner field to represent the current thread, and grants access to the newly locked object. If the lock field is non-NULL, enterMonitor examines the contents of the HashLock object to determine whether access to the lock can be granted (*Nilsen et al.,* col. 35, lines 30-35). However, it is respectfully submitted that the object lock field of *Nilsen et al.* does NOT teach or remotely suggest a virtual machine arranged to enable two or more jobs, which can be associated with two or applications, to run thereon.

***(b) Nilsen et al.* does NOT teach or suggest a virtual machine arranged to create a heap in the memory for each job that runs on the virtual machine (claim 1)**

It is noted that *Nilsen et al.* states that it is necessary to provide parameterized access to **a** heap memory so as to facilitate implementation of read and write barriers. (*Nilsen et al.*, Col. 20, lines 21-44.) Contrary to the Examiner's assertion, however, it is respectfully submitted that parameterized access to a heap memory described by

Nilsen et al. does not teach or suggest creating a heap in the memory for each job that runs on the virtual machine. Moreover, *Nilsen et al.* does NOT teach or even remotely suggest generating a heap for each job. In fact, *Nilsen et al.* teaches away from generating more than one heap because it teaches providing parameterized access to a single heap memory.

It is further noted that *Nilsen et al.* states:

> When the pvm( ) (PERC Virtual Machine byte code interpreter) is executing byte-code methods, the method's byte code is represented by a string of bytes. The byte-code instructions are stored in heap memory, suggesting that every instruction fetch needs to incur the overhead of a heap-access macro. To improve the performance of instruction fetching, we allow instruction fetching to bypass the standard heap access macro (*Nilsen et al.*, col. 25, lines 5-12).

Contrary to the Examiner's assertion, it is respectfully submitted that instruction fetching of *Nilsen et al.* which bypasses the standard heap access does NOT teach or suggest creating a heap in the memory for each job that runs on the virtual machine

(c) *Nilsen et al.* does NOT teach or suggest a virtual machine that includes a heap manager that manages all heaps in the memory that are created for two or more jobs (claim 4)

It is noted that *Nilsen et al.* in section 5.1 describes parameterized access to a heap memory (*Nilsen et al.*, col. 20, line 15 to col. 21 line 44). However, contrary to the Examiner's assertion, it is respectfully submitted that parameterized access to a heap memory as described by *Nilsen et al.* does NOT teach or remotely suggest this claimed feature. Moreover, it is respectfully submitted that *Nilsen et al.* cannot possibly teach or suggest this feature because of the general deficiencies discussed above

***(d) Nilsen et al.* does NOT teach or suggest a virtual machine that includes a heap manager arranged to allow an object allocated on a first heap to be visible to a second heap (claim 5)**

Again, it is very respectfully submitted that parameterized access to a heap memory as described by *Nilsen et al.* (*Nilsen et al.*, col. 20, line 15 to col. 21 line 44) does NOT teach or remotely suggest this claimed feature. Moreover, it is respectfully submitted that *Nilsen et al.* cannot possibly teach or suggest this feature because of the general deficiencies discussed above.

***(e) Nilsen et al.* does NOT teach or suggest a virtual machine further arranged to create an incremental garbage collector for each of the heaps (claim 11)**

It is noted that Fig. 2 of *Nilsen et al.* illustrates a header of information attached to each dynamically allocated memory object for purposes of performing garbage collection. These header fields consist of Scan-List, Indirect-Pointer, Activity-Pointer, Signature-Pointer and optional Finalize-Link pointers (*Nilsen et al.*, col. 6, lines 40-45). It is also noted that Fig. 32 of *Nilsen et al.* illustrates a signature structures used to represent a memory layout of heap allocated objects (*Nilsen et al.*, col. 8, lines 46-47). However, it is respectfully submitted that neither of the structures shown in Fig. 2 and 32 of *Nilsen et al.* teach or even remotely suggest a virtual machine further arranged to create an incremental garbage collector for each of the heaps.

In addition, it is further noted that *Nilsen et al.* states that in some cases, such as when a dynamically allocated object contains union fields that contain pointers only some of the time, it is necessary to allocate a private copy of the signature along with the actual object. To minimize allocation overhead, both the signature and the data are allocated as a single contiguous region of memory using an allocation routine of *Nilsen et al.* (*Nilsen et al.*, col. 45, lines 39-46). However, it is respectfully that the allocation routine described by *Nilsen et al.* does NOT teach or suggest this feature. Moreover, it is respectfully submitted that *Nilsen et al.* cannot possibly teach or suggest this feature because of the general deficiencies discussed above.

Conclusion

Based on the foregoing, it is further submitted that claims are patentably distinct over the cited art of record. Additional limitations recited in the independent claims or the dependent claims are not further discussed because the limitations discussed above are sufficient to distinguish the claimed invention from the cited art. Accordingly, Applicant believes that all pending claims are allowable and respectfully requests a Notice of Allowance for this application from the Examiner.

Applicants hereby petition for an extension of time which may be required to maintain the pendency of this case, and any required fee for such extension or any further fee required in connection with the filing of this Amendment is to be charged to Deposit Account No. 500388 (Order No. SUN1P270). Should the Examiner believe that a telephone conference would expedite the prosecution of this application, the undersigned can be reached at the telephone number set out below.

Respectfully submitted,
BEYER WEAVER & THOMAS, LLP

R. Mahboubian
Reg. No. 44,890

P.O. Box 778
Berkeley, CA 94704-0778
(650) 961-8300